2020년도 2학기 컴퓨터공학설계및실험Ⅰ

9주차 테트리스 3주차 예비보고서

20161663 허재성

1. 실습 목적

테트리스 게임을 플레이하는 사용자를 위해 어떤 위치에 블록을 놓으면 높은 점수를 받을 수 있는지를 추천하는 추천 시스템을 구현한다. 추천된 블록은 1주차 숙제의 그림자 기능을 이용해서 화면에 나타낸다. 이 과정을 통해서 tree와 포인터(pointer)에 대한 이해를 높일 수 있다.

2. 관련 이론

2-1. 교재를 참조하여 테트리스 프로젝트 3주차에 구현하는 추천 기능은 어떤 원리로 작동하는지설명하시오. 추천 기능을 구현하는 tree 구조의 장점(효율성)과 단점(비효율성)을 기술하시오.

테트리스 1주차 실습과 과제에서 현재 블록과 다음에 나올 2개의 블록, 즉 다음 블록, 다다음 블록을 미리 보여주는 기능을 구현하였다. 앞으로 나올 2개의 블록을 미리 알 수 있으면, 현재 블록을 어떤 위치에 놓았을 때 장기적으로 더 큰 점수를 얻을 수 있는 지 예상할 수 있다.

테트리스 블록의 종류는 총 7가 있고, 중복되는 모양을 별개의 모양으로 생각하면 모든 블록은 각각 4개의 회전 상태를 가지고 있다. 예를 들어, 테트리스 블록 0(흔히 말하는 작대기 모양 블록)은 회전 상태 0, 2일 경우 누워 있는 모양으로 같고, 1, 3일 경우 서 있는 모양으로 같지만 서로 다른 경우로 간주한다. 테트리스 블록 4(정사각형 모양 블록)는 회전 상태 0, 1, 2, 3 모두 정사각형 모양으로 같지만 모두 다른 경우로 생각한다.

테트리스 필드에서 블록의 좌표는 각 블록을 표현하는 4\*4 배열의 왼쪽 위 좌표에 대응한다. 예를 들어 필드의 (5, 6)에 블록이 있다면, 블록을 표현하는 배열의 왼쪽 위 좌표 (0, 0)가 필드의 (5, 6)에 대응된다.

각 블록의 회전 상태에 따른 존재 가능한 필드의 x 좌표의 범위를 표로 정리했다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **회전 상태 0** | **회전 상태 1** | **회전 상태 2** | **회전 상태 3** |
| **블록 0** | 0 ≤ x ≤ 6 | -1 ≤ x ≤ 8 | 0 ≤ x ≤ 6 | -1 ≤ x ≤ 8 |
| **블록 1** | -1 ≤ x ≤ 6 | -2 ≤ x ≤ 6 | -1 ≤ x ≤ 6 | -1 ≤ x ≤ 7 |
| **블록 2** | -1 ≤ x ≤ 6 | -2 ≤ x ≤ 6 | -1 ≤ x ≤ 6 | -1 ≤ x ≤ 6 |
| **블록 3** | 0 ≤ x ≤ 7 | 0 ≤ x ≤ 8 | 0 ≤ x ≤ 7 | -1 ≤ x ≤ 7 |
| **블록 4** | -1 ≤ x ≤ 7 | -1 ≤ x ≤ 7 | -1 ≤ x ≤ 7 | -1 ≤ x ≤ 7 |
| **블록 5** | -1 ≤ x ≤ 6 | -1 ≤ x ≤ 7 | -1 ≤ x ≤ 6 | -1 ≤ x ≤ 7 |
| **블록 6** | -1 ≤ x ≤ 6 | -1 ≤ x ≤ 7 | -1 ≤ x ≤ 6 | -1 ≤ x ≤ 7 |

블록 4의 경우 x 좌표 -1부터 x 좌표 7까지 총 9개의 위치에 존재 가능하다. 모든 회전 상태에서 모양이 같으므로 블록 4의 경우, 회전 상태에 따라 총 36개의 위치에 존재가 가능하다. 블록 4의 경우가 가장 많은 위치를 가진다. 나머지 블록들은 34개의 위치가 가능하다.

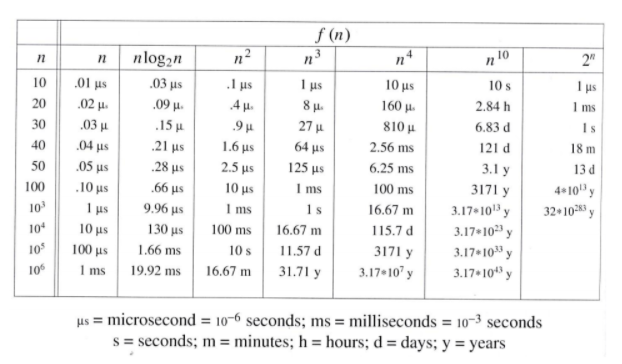
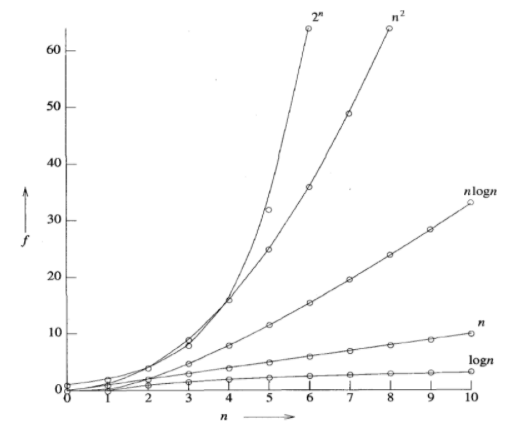
x 좌표의 경우 블록 2와 블록 3이 회전 상태 2에서 -2에 위치할 수 있다. 최대 8까지 이동할 수 있는 블록들이 있으므로 블록이 존재 가능한 x 좌표의 최대 범위는 -2 ≤ x ≤ 8로 생각할 수 있다. y 좌표의 경우, 해당 회전 상태와 x 좌표의 위치에서 최대로 내린 경우의 y 좌표이므로 고려하지 않는다.

현재 블록이 위치했을 때, 다음 블록(첫 번째 다음 블록)이 놓일 수 있는 위치의 경우의 수는 최대 36가지이다. 다음 블록의 각각의 위치(최대 36개의 위치)에 대해서, 그 다음 블록(두 번째 다음 블록)이 놓일 수 있는 위치의 경우의 수도 최대 36개이다. 트리의 구조를 생각해보면 최대 36개의 자식 노드를 가질 수 있는 36진 트리를 생각할 수 있다. 트리의 root 노드의 level을 0으로 설정할 경우, 현재 블록의 위치를 나타내는 최대 36개의 자식 노드들의 level은 1이 된다. Level 1의 각 노드들이 나타내는 위치에 대하여 다음 블록의 위치는 최대 36개가 가능하므로 level 1의 노드 당 자식 노드도 최대 36개이므로 level 2의 노드의 개수는 362 = 1,296개가 된다. 이와 같은 방식으로 계산해보면 level N의 노드의 개수는 최대 36N개가 된다.

테트리스 프로젝트 3주차의 추천 기능에서는 현재 블록, 첫 번째 다음 블록, 두 번째 다음 블록 총 3개의 블록으로 최대 점수를 예측한다. 첫 번째 블록을 놓을 수 있는 36가지의 위치(블록의 모양, 회전 상태를 고려한 위치의 개수)의 노드에 대하여 얻을 수 있는 점수를 계산 후 해당 노드에 저장한다. 해당 노드의 자식 노드 최대 36개를 고려하여 해당 위치에 놓았을 때 얻을 수 있는 최대 누적 점수를 계산한다. 이 때, 해당 노드를 먼저 탐색하고, 자식 노드들을 차례로 탐색한다. 모든 노드를 탐색하여 누적 점수를 계산 후 비교해서 가장 많은 점수를 가지고 있는 자식 노드를 찾아 추천 블록의 위치를 정한다.[1]

트리 구조를 사용하여 현재 블록이 놓일 수 있는 36가지의 경우의 수와, 현재 블록의 각각의 위치에 대하여 다음 블록의 위치 36가지, 이러한 방식으로 모든 경우의 수를 전부 고려한다. 모든 경우의 수를 고려할 수 있으므로 주어진 조건에서 최선의 선택이 가능하다. 고려하는 블록의 개수가 많아질수록, 더 많은 경우의 수를 따져서 더 큰 점수를 얻는 방법을 찾을 수 있다. 트리 자료구조를 사용하여 모든 경우의 수를 고려한다는 막연한 개념을 구체적으로 구현할 수 있다.[1]

트리 자료구조를 사용하여 모든 경우의 수를 고려할 경우, 블록이 하나 증가할 때마다고려해야 할 경우의 수는 기하급수적으로(expotentially) 증가한다. 만약 현재 블록과 다음 블록 하나만 고려한다면 고려할 경우의 수는 36+362=1332개이다. 두 번째 다음 블록까지 고려한다면 경우의 수는 36+362+363=47,988개이다. 현재 블록까지 총 N개의 블록을 고려하면, (36N+1-1)/35개의 경우의 수를 고려해야 한다. 고려할 블록의 개수가 1개 증가할 때 고려해야 할 경우의 수는 대략 36배 증가한다. 따라서 모든 경우의 수를 따지는 시간복잡도는 O(36n)이다.[1]



시간 복잡도가 O(n), O(nlog2n), O(n2), O(n3), O(n4), O(n10), O(2n)인 함수들의 수행 시간을 비교한 그래프와 표이다.[2] 시간 복잡도가 2n인 경우 n이 조금만 증가하여도 현실적으로 불가능한 시간이 소요된다. 36 > 25이므로 n=10만 되어도 250만큼을 처리하는 시간(표에서는 13일)보다 더 오랜 시간이 걸린다.

트리의 노드마다 자식 노드가 최대 36개씩 존재할 수 있으므로 공간 복잡도 또한 O(36n)이다. 각 노드가 차지하는 공간을 생각해보면 현재 노드의 level과 점수, 필드의 정보 22\*10을 저장하고, 또한 자식 노드 36개의 포인터를 저장해야 한다. 하나의 노드가 차지하는 공간이 적지 않다. 하나의 노드가 차지하는 공간을 상수 c라고 하면, 트리가 차지하는 공간은 노드의 수 \* c이다. 블록의 수가 1 증가할 때, 노드의 수가 36배으로 증가하므로 차지하는 공간 또한 36배 증가한다. 따라서 공간 복잡도도 O(36n)이다.[1]

2-2. Tree 구조의 비효율성을 해결할 방법에 대해서 2가지 이상 생각하고, 그 idea를 기술하시오.

각 노드가 가질 수 있는 자식 노드의 개수를 줄이면 시간 복잡도를 크게 줄일 수 있다. 블록이 회전했을 때 같은 모양이 되는 회전 상태를 서로 다르게 생각했을 때, 최대 경우의 수가 36이였다. 회전하여 같은 모양이 되는 경우를 같은 경우로 생각한다. 블록 0의 경우, 회전 상태 0과 2는 서로 모양이 같고, 1과 3도 서로 모양이 같다. 따라서 가능한 회전의 수를 4가 아닌 2(0, 1)로 생각할 수 있고 이 경우, 블록의 가능한 위치는 17개이다. 블록 1, 2, 3의 경우 4가지 회전의 경우가 모두 다르므로 기존과 같이 4가지 회전 상태, 34개의 위치를 가진다. 블록 4의 경우 4가지 회전의 경우가 모두 같으므로 회전 상태는 1가지를 가진다. 따라서 9개의 위치를 가진다. 블록 5, 6의 경우 회전 상태 0, 2가 같고, 1, 3이 같으므로 회전 상태는 0, 1 2가지만 가지고 17개의 위치를 가진다. 이렇게 회전 상태를 제한할 경우 자식 노드의 개수를 줄일 수 있다. 예를 들어 다음 블록이 블록 4일 경우 블록 4는 회전 상태가 한 가지이므로 블록 4가 놓일 수 있는 위치는 9가지이다. 현재 블록 입장에서 다음 블록이 블록 4일 경우, 36개의 자식 노드가 아닌 9개의 자식 노드만 동적 할당하면 된다. 다음 블록이 정해 졌을 때, 해당 블록이 회전 경우의 수의 중복을 고려하여 최대 위치의 가짓수를 생각하면 자식 노드의 개수를 최소화할 수 있다.

모든 블록의 위치 가능한 x 좌표의 범위를 획일적으로 -2 ≤ x ≤ 8로 설정하여 검사하지 않고, 블록의 종류에 따라 알맞은 범위를 검사한다. 이 방법만으로 할당되는 노드의 개수를 줄일 수는 없지만, 하나의 노드에 대하여 다음 블록의 위치를 고려하는 시간을 감소시킬 수 있다.

트리 노드의 자식 블록을 가리키는 포인터 배열을 다음 블록의 종류에 따라 동적 할당한다. 즉, 다음 블록의 종류에 관계없이 최대 36개의 자식 노드를 가질 것을 예상하여 포인터 배열을 정적 할당하였다면, 다음 블록의 종류에 따라 필요한 자식 노드의 수를 결정하여 포인터 배열을 동적할당할 수 있다. 하나의 노드에서 이로 인한 공간 절약은 크지 않을 수 있지만, 할당해야 하는 노드의 수가 많을 경우 이와 같은 방법으로 적지 않은 공간을 절약할 수 있다.

가지 치기(Pruning) 방법을 사용하여 고려해야 할 노드의 개수를 줄일 수 있다. 예를 들어 특정 노드의 자식 노드들이 있을 때, 각 자식 노드들의 점수를 비교하여 점수가 현저히 낮은 노드와 해당 노드의 자식 노드들에 대해서는 고려하지 않는다. 이와 같은 방법으로 고려해야 할 경우의 수가 줄어든다. 예를 들어 최대 level이 N인 36진 트리에서 root 노드 기준으로 하나의 자식 노드를 가지 치기하면 고려해야 할 노드의 개수가 1+36+362+…+36N-1=(36N-1)/35개 감소한다. 약 1/36이 감소한다. 만약 가장 점수가 높은 자식 노드만 남기고 나머지 자식 노드를 전부 가지치기한다면 고려해야 할 노드의 개수는 약 1/36로 감소하여 시간 복잡도를 획기적으로 감소할 수 있다. 하지만 가지 치기 방법의 경우 가장 점수가 큰 자식 노드를 선택했다고 해서 해당 위치에 블록을 놓는 것이 꼭 최고 점수를 보장하지 않는다. 자식 노드의 점수가 낮아서 가지치기한 경우에서 더 큰 점수의 경우가 발생할 수도 있다. 첫 번째 다음 블록만 고려하여 당장 가장 큰 점수의 노드를 선택할 경우, 그 다음 블록을 고려했을 경우의 최고 점수를 보장할 수 없다.[1] 이와 같이 그리디 알고리즘을 활용하여 가지 치기를 하는 경우 그 순간에 최적의 선택을 할 수는 있지만 최종적인 선택이 최적의 선택임을 보장하지 못하고, 근사적인 방법으로 사용된다.[3]

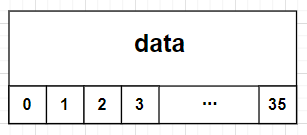
마지막으로 노드에 필드의 정보를 전부 저장하는 방법이 아닌, 필드의 각 x좌표에 대하여 블록의 최대 높이만을 저장하는 방법을 고려할 수 있다. 필드의 x좌표(0~WIDTH-1)마다 쌓여 있는 블록의 최대 높이만을 계산하여 저장하면 노드 하나가 차지하는 필드 관련 공간을 O(HEIGHT\*WIDTH)에서 O(WIDTH)로 줄일 수 있다. 하지만 필드의 블록의 최대 높이만을 계산하기 때문에 필드에 쌓여 있는 블록들의 내부의 빈 공간은 고려되지 못하고, 이로 인해 잘못된 점수 계산이 이루어져서 잘못된 예측이 이루어질 수 있다.[1]

3. 실습 방법

3-1. 실습 시 자신이 사용할 Tree 자료구조에 대해 소개하시오.

실습에서는 트리의 각 노드가 최대 36개의 자식 노드를 가질 수 있는 36진 트리를 사용한다.

사용할 36진 트리의 노드는 RecNode로 그림으로 간략히 표현하면 다음과 같다.



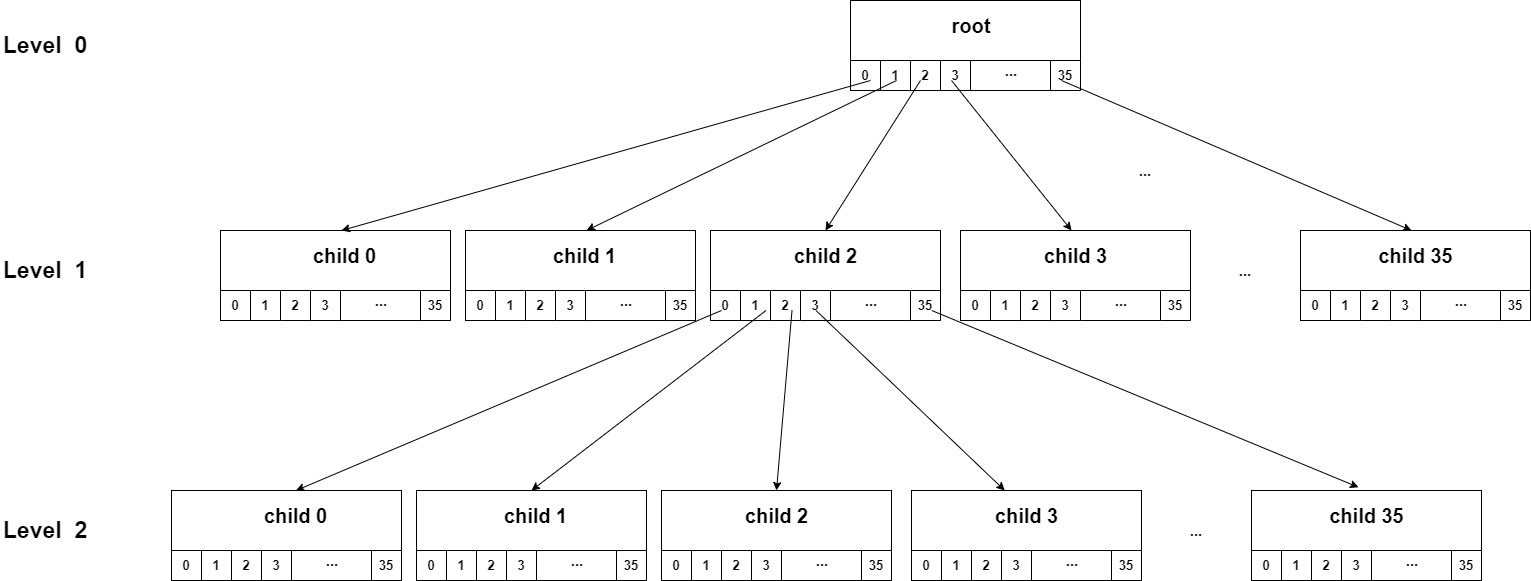
구조체 RecNode는 다음과 같이 정의되어 있다.[1]

|  |
| --- |
| typedef struct \_RecNode {  int lv, score;  char (\*f)[WIDTH];  struct \_RecNode\* c[CHIlDREN\_MAX];  } RecNode; |

구조체의 멤버들에 대한 설명은 다음과 같다.[1]

|  |
| --- |
| int lv : 트리에서 현재 노드의 level(depth)를 나타낸다. root 노드의 level=0을 기준으로 한다.  int score : 현재 level까지 블록을 놓았을 때 누적 점수이다.  char (\*)[WIDTH] : 블록을 테트리스 필드에 놓았을 때의 필드 상태를 나타내는 배열을 가리키기 위한 배열 포인터이다.  struct \_RecNode\* c[CHILDREN\_MAX] : 자식 노드들을 가리키는 포인터 배열이다. CHILDREN\_MAX는 36으로 tetris.h 파일에 선언되어 있다. 최대 36개의 자식 노드를 가리킬 수 있다. |

위의 RecNode를 사용해 구현한 트리의 모양은 대략적으로 다음과 같다.



root 노드의 level을 0으로 하고 기준으로 삼는다. Level 1에 존재 가능한 최대 36개의 노드 중 일부만을 표현하였다. Level 2에 존재 가능한 최대 362개의 노드 중, level 1의 child2 노드의 자식 노드 36개, 그 중에서도 일부만 표현하였다.

트리의 level이 N일 경우, 노드의 개수는 최대 1 + 36 + 362 + … + 36N = (36N+1-1) / (36 – 1)이다.

root 노드를 제외할 경우 (36N+1-36) / (36 – 1)개이다.

3-2. 추천 기능의 구현을 의사 코드(Pseudo Code)로 작성한 후, 시간 및 공간 복잡도를 계산하시오.

|  |
| --- |
| int recommend(RecNode\* root) {  int max = 0; // 최대 누적 점수  int temp; // 각 경우의 누적 점수  for(모든 가능한 회전 수) {  for(모든 가능한 x의 위치) {  if(블록이 놓일 수 없을 경우) continue;  자식 노드 child 할당;  블록이 놓인 위치에 맞게 자식 노드의 필드 정보를 업데이트하여 저장;  child->lv = root->lv + 1;    child->score += 현재 블록을 필드에 놓았을 때 얻는 점수;  child를 root의 자식 노드로 함;  if(현재 level < 최대 블록의 수) temp = recommend(child);    if(max < temp) {  max = temp;  추천 위치 결정;  }  }  if(root->lv == 0)  최종적인 추천 위치 결정;  return max;  } |

점수 계산을 위해 이전에 정의했던 AddBlockToField, Delete 함수를 이용한다.

N개의 블록을 고려할 때, 최대 O(36N)개의 노드가 할당된다. 따라서 공간 복잡도는 O(36N)이다. 할당된 노드를 모두 고려해야 하므로 시간 복잡도는 O(36N)이다.

4. 기타

모든 경우의 수를 예측하기 위해서는 expotential한 시간 복잡도와 공간 복잡도를 가지는 것은 불가피하다. 트리를 구축하고 탐색하는 데 이진 트리의 preorder traversal[2] 방법과 유사하게 루트 노드를 먼저 탐색해 루트 노드의 점수를 계산 후 자식 노드를 할당 후 재귀적으로 탐색한다. 사용한 트리를 해제하기 위해서는 반대로 postorder traversal[2]의 방법을 응용한다. 자식 노드들을 모두 재귀적으로 삭제하고, root 노드를 삭제한다.

[1] 서강대학교 컴퓨터공학과 교수진, “컴퓨터공학 설계 및 실험 I”, *서강대학교 공학부 컴퓨터공학과*, 2012

[2] [Ellis Horowitz](http://www.yes24.com/SearchCorner/Result?domain=ALL&author_yn=Y&query=Ellis+Horowitz), [Sartaj Sahni](http://www.yes24.com/SearchCorner/Result?domain=ALL&author_yn=Y&query=Sartaj+Sahni), [Susan Anderson-Freed](http://www.yes24.com/SearchCorner/Result?domain=ALL&author_yn=Y&query=Susan+Anderson-Freed), “Fundamentals of Data Structures in C”, [*Silicon Valley Publishers Group*](javascript:void(0);), 2007

[3] 위키백과 “탐욕 알고리즘”

https://ko.wikipedia.org/wiki/%ED%83%90%EC%9A%95\_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98